	Politecnico di Milano Facoltà di Ingegneria dell'Informazione **Sistemi Informatici** Appello 11 febbraio 2011		COGNOME E NOME
	RIGA	COLONNA	MATRICOLA

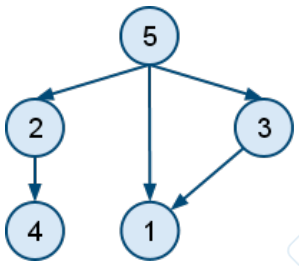
- Il presente plico pinzato, composto di quattro fogli (fronte/retro), deve essere debitamente compilato con cognome, nome, numero di matricola, posizione durante lo scritto, e deve essere firmato.
- I compiti non compilati, non firmati o con fogli mancanti non saranno considerati validi e quindi non saranno corretti.
- Sarà valutato solo quanto scritto su questi fogli.
- Non è consentito consultare testi né appunti.
- Sul tavolo non devono essere presenti telefoni cellulari, né astucci, né custodie di altro tipo.

FIRMA

Esercizio 1 (10 punti). Si modelli tramite il formalismo degli StateCharts un problema di gestione di revisioni condiviso. Si hanno due utenti, A e B, che lavorano su di un documento condiviso organizzato in due sezioni, S1 e S2. Ogni utente può lavorare in una qualunque sezione. Ogni volta che un utente accede ad una sezione, la sezione viene caricata (stato *in_caricamento*). Quando l'utente inizia a scrivere sulla sezione (evento *scrittura*) lo stato passa a *in_modifica*. A questo punto l'utente può sottomettere la sua modifica al sistema di revisioni (evento *sottometti*). Se la versione sottomessa non genera conflitti con la versione precedentemente caricata sul sistema di revisioni (vedi sotto), allora si passa a *salva* e successivamente, dopo un *timeout*, a *in_caricamento*. Se la versione sottomessa genera conflitti con la versione precedentemente caricata sul sistema di revisioni (vedi sotto), allora si passa a *gestione_conflitti* e una volta risolti, *conflitti_risolti*, a *salva*. Per poter gestire il riconoscimento dei conflitti si usino due stati per sezione per utente (ad esempio, *aggiornataA1* e *non_aggiornataA1*) che in ogni istante di tempo segnalano se la versione correntemente in uso ad un utente è la versione che è presente sul sistema di revisione. Un utente può passare da una sezione ad un altra tramite un evento *Asezione1* oppure *Asezione2* (lo stesso per l'utente B). Il sistema è fatto in modo tale che se un utente cambia sezione su cui sta lavorando, non perde le modifiche fatte precedentemente (anche se non sono state salvate).

Esercizio 2 (10 punti). Siano dati i seguenti processi aperiodici J_1 - J_5 riportati in tabella e i rispettivi vincoli di precedenza:

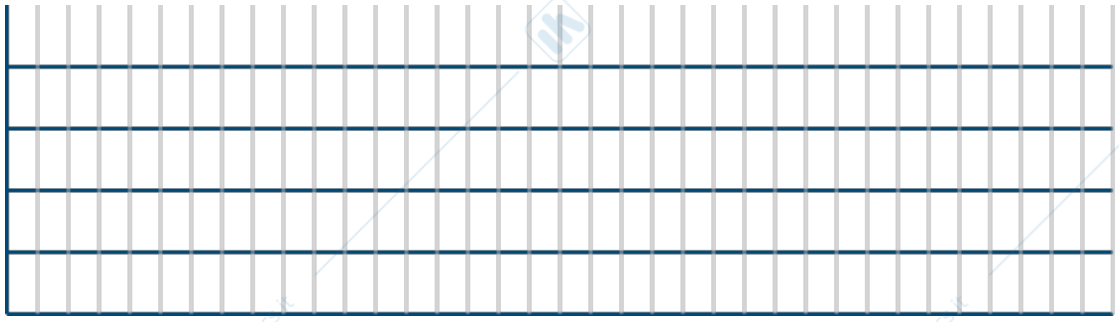
	J_1	J_2	J_3	J_4	J_5
a_i	2	1	0	1	3
C_i	2	1	3	2	1
d_i	11	10	9	12	14



1. Si applichi l'algoritmo EDF* riportando i valori di a^* e d^* e si dica, riportando la schedulazione e il valore di L_{max}, se il problema è schedulabile.

2. Si consideri la tabella iniziale dei processi e si assuma di non avere la possibilità di effettuare preemption. (Si assuma però che tutti i dati siano noti all'istante iniziale.) Si discuta come affrontare la schedulazione di questo problema e si applichi una algoritmo appropriato per generare una soluzione non necessariamente ammissibile.

3. Si consideri una variante dell'algoritmo TBS in cui la deadline di una richiesta aperiodica J_i viene assegnata considerando la deadline assegnata a J_{i-2} invece della deadline assegnata a J_{i-1}. Si risponda alle seguenti domande motivando le risposte:
3.1 L'introduzione di questa modifica porta a un miglioramento o peggioramento dei tempi di risposta delle richieste aperiodiche?
3.2 Qual'è il fattore di utilizzazione massimo del server che garantisca la schedulabilità dei processi? Si noti che il caso peggiore (similmente a quando avviene con la sezione critica per RM) corrisponde alla situazione in cui si hanno tre richieste aperiodiche J_1 J_2 J_3 tali che: $a_2 = a_3 = d_1$ e $C_2 = C_3$.



Esercizio 3 (6 punti). Si presenti sinteticamente il problema della sincronizzazione tra processi concorrenti e si descrivano, a tratti generali, il funzionamento e le caratteristiche di un monitor.

Esercizio 4 (6 punti). Si consideri il frammento di codice Java riportato sotto e si risponda alle seguenti domande.

1. Si dica, motivando la risposta, se è sempre possibile o non possibile (indipendentemente dall'implementazione della classe) definire un attributo di ClasseA all'interno di una ClasseA.
2. Rappresentare la memoria alla terminazione di ognuna delle 5 istruzioni del main.

```
public class ClasseA {
        public static int n;
        private int val;
        private ClassA x;

        public ClasseA () {
                val = 0;
                n = 16;
                x = this;
        }

        public ClasseA (ClasseA x) {
                this.x = x;
                val = x.getVal () + 1;
                ClasseA.n = ClasseA.n/2;
        }

        public int getVal () {
                return val;
        }

        public ClasseA getX () {
                if (val == 0)
                        return this.x;
                else
                        return x.getX ();
        }

        public static void main (String[] args) {
                ClasseA o1, o2, o3, o4;          //1
                o1 = new ClasseA ();             //2
                o2 = new ClasseA (o1);           //3
                o3 = new ClasseA (o2);           //4
                o4 = o3.getX ();                 // 5
        }
} // end ClassA
```